

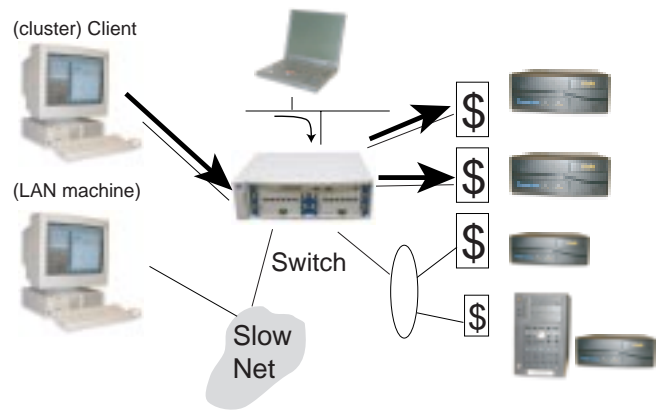
Dynamic Function Placement for Data-Intensive Cluster Computing

{Khalil.Amiri, David.Petrou, Greg.Ganger, Garth.Gibson}@cs.cmu.edu

<http://www.pdl.cs.cmu.edu>

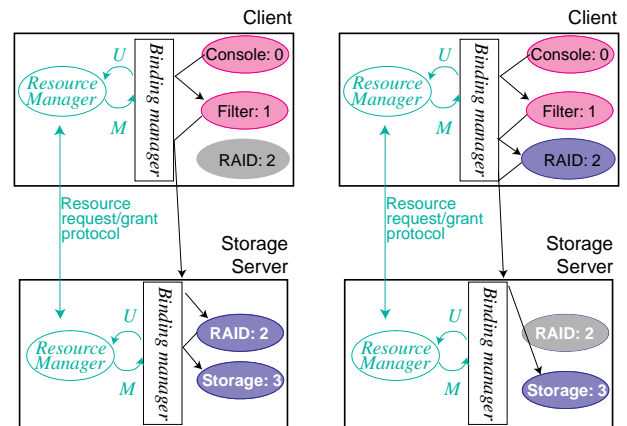
Motivation

- Feasibility of active storage clusters
- Heterogeneity of node (client/server) resources
 - Variability in network links between them
- Variability in workloads and in workload mixes
 - Competing and multiphasic applications
- Dynamic placement of function is crucial
 - Best placement depends on workload and topology
 - These characteristics change dynamically
 - Continual automatic reconfiguration



Approach

- Define an object-based programming model
 - Object-based applications and filesystems
- System continuously monitors objects
 - Object resource consumption (memory, CPU)
 - Object communication (timed data flow graph of bytes moved)
- System predicts "optimal" placement using on-line analytic model
 - Uses the resource statistics collected by the run-time system



ABACUS Prototype

- Programming model
 - Explicitly migratable "mobile objects"
 - Iterative block-based processing
- Run-time system
 - Instrument procedure calls/returns from mobile objects
 - Transparently redirect invocations
 - Maintain statistics about bytes moved, instructions executed, memory used
- Dynamic object placement subsystem
 - Clients monitor object resource consumption and communication
 - Clients summarize stats and send periodic updates to the server
 - Server collects and buffers statistics
 - Server drives placement decisions (see below)

On-Line Placement Algorithm

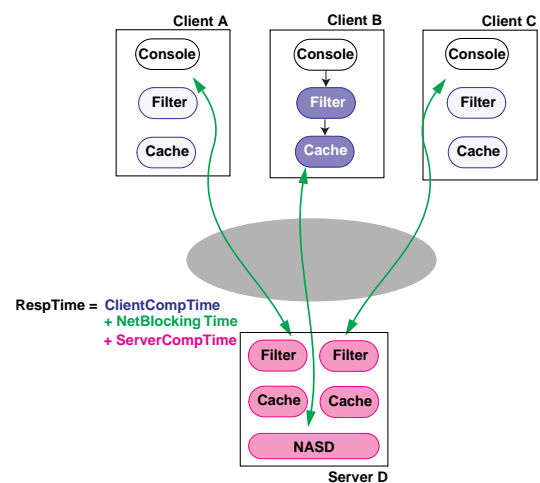
Goal: Minimize average application execution (response) time.

Cost-benefit model

- Allocate excess server resources to clients objects
- Benefit: reduction in execution time
- Cost: migration penalty

Assumptions

- Past history is a good predictor of future behaviour
- "Processor sharing" of server's CPU among client objects
- Server memory is a hard constraint



Ongoing Work

- How to best decompose applications into objects to run on ABACUS?
- How to extend ABACUS beyond just client and server (e.g., to active network nodes)?



ABACUS: Early Results

{Khalil.Amiri, David.Petrou, Greg.Ganger, Garth.Gibson}@cs.cmu.edu

http://www.pdl.cs.cmu.edu

Evaluation Environment

Eight clients and four active storage servers

- All workstations are RedHat Linux 300 MHz Pentium IIs

Two networks

- 100 Mbps switched Ethernet (fast net)
- 10 Mbps Ethernet (slow net)

Variable bandwidth between clients and servers

- Active storage servers connected to the SAN
- 4 clients on the fast net and 4 clients on the slow net

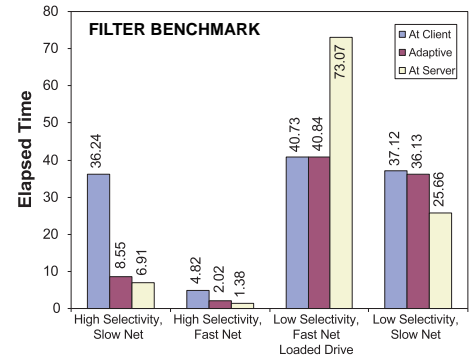
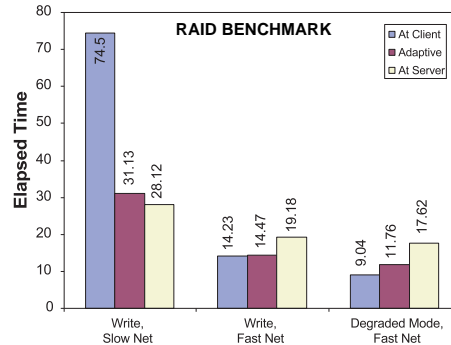
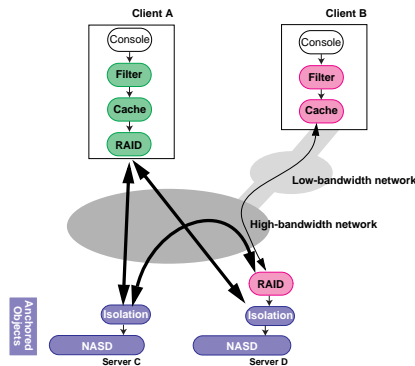
Function Placement Depends on Workload & System Characteristics

• Workload characteristics

- Bytes moved, instructions executed, selectivity

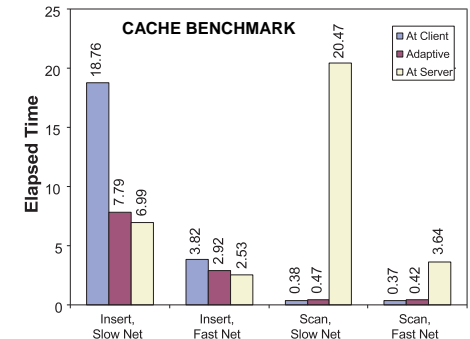
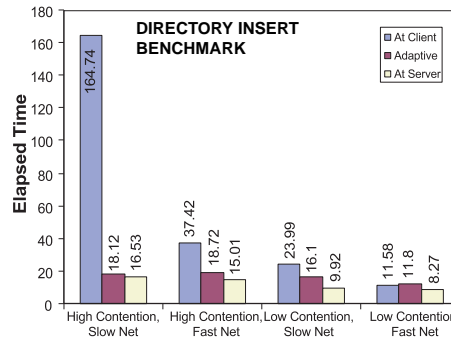
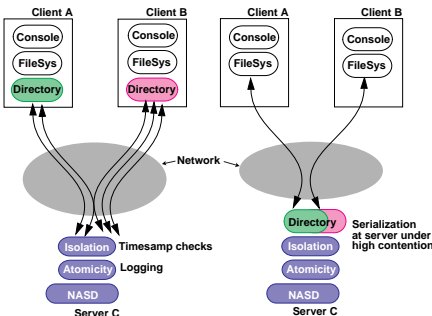
• System characteristics

- Available bandwidth between client and server
- Available resources at the server



- RAID small writes at server when bandwidth is scarce
- RAID small writes at client when server is loaded

- Filtering at server when network bandwidth is limited
- Filtering at client when high instr./byte or loaded server



- Directory ops at clients under low contention
- Directory ops at server under high contention

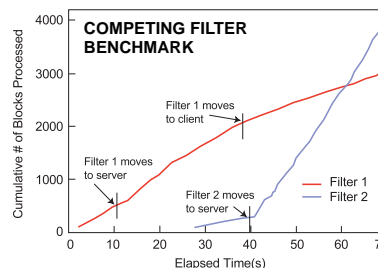
- Cache at client if application exhibits reuse
- Cache at server if small updates induce too many installation reads

Relevant Characteristics Change Dynamically

Relevant characteristics change at run-time

- Available bandwidth to server fluctuates
- Application may have multiple phases
- Application mix changes dynamically

	Insert	Scan	Insert	Scan	Total
At Client	26.03	0.41	28.33	0.39	55.16
Adaptive	11.69	7.22	12.15	3.46	34.52
At Server	7.76	29.2	7.74	26.03	70.73
Optimal	7.76	0.41	7.74	0.39	16.3



- Continuous run-time monitoring adapts to application phases
- Placement algorithm resolves competition over server resources



Concurrency Control and Recovery for Shared Storage Arrays

{Khalil.Amiri, Garth.Gibson}@cs.cmu.edu, Richard.Golding@hpl.hp.com
<http://www.pdl.cs.cmu.edu/>

Shared Storage Arrays

Scalable networks enable wide device sharing,
 scalable single-system storage

When there is no single storage controller

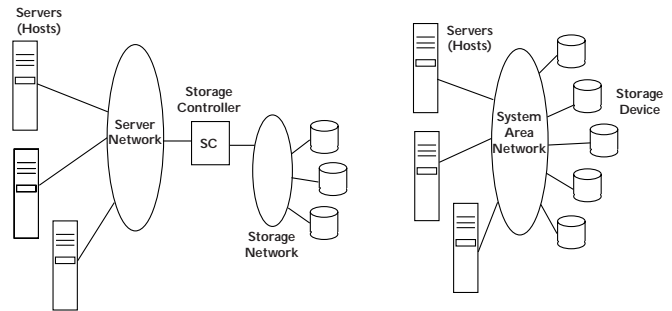
- Consistency of redundancy codes in jeopardy
- Host accesses can be interleaved

Ensure isolation for host ops, consistency

- Consistency after failures, not atomicity
- Isolation of hosts' requests (serializability)

Approach

- Distribute work to devices to improve scaling
- Combine IOs with lock/ordering messages to reduce latency



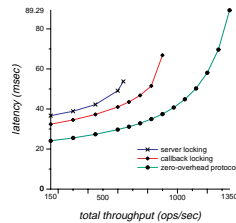
Centralized Locking

Limited scalability

- Lock server bottlenecks on message request processing

Callback locking suspect

- If locality is lacking
- Due to false sharing introduced by storage layer



Distributed Protocols

Device-based locking

- Avoids bottlenecks, load distributed across devices
- Pre-I/O latency reduced, vulnerability reduced
- More messaging, deadlocks become possible

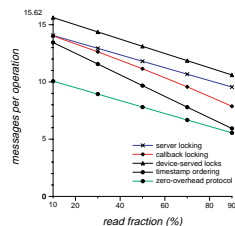
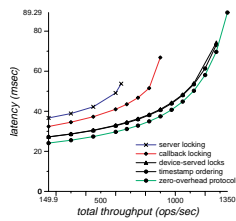
Timestamp ordering

- Host selected from local clock
- Devices enforce ordering
- Free from deadlocks, no messaging on reads
- Pre-I/O latency reduced with piggy-backing

Scalability

Distributed device-based protocol scales well

- 90-95% of Maximum possible throughput
- Timestamp ordering has low messaging

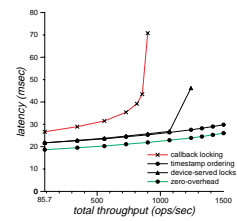
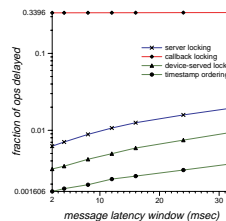


Message Skew, Disk Speed

Skew not a problem for distributed protocols

Device-based protocols continue to scale

- Timestamp ordering favorable under high load/contention



Consistency on Failure Recovery

- Locking requires durable locks to detect suspect ranges; induces extra IOs
- Timestamp ordering with linkage records
 - Description of high-level write embedded with data
 - Consistency without extra IOs

